PyWSD Documentation

Patrick Roncagliolo

Feb 26, 2020

Contents

1 Structures **3**
1.1 Discovery . . . 3
1.2 Transfer . . . 3
1.3 Scan . . . 3

2 Operations **5**
2.1 Discovery . . . 5
2.2 Transfer . . . 7
2.3 Eventing . . . 7
2.4 Scan . . . 8

Python Module Index **13**

Index **15**

PyWSD is a library for interacting with WSD-enabled devices in your network. WSD stands for Web Services for Devices, a set of technologies based on the exchange of XML messages between devices such as a PC and a network printer, but in fact the scope of application is theoretically broader than just printers and scanners: you could query a WSD-enabled thermometer, for example.

CHAPTER 1

Structures

1.1 Discovery

class PyWSD.wsd_discovery__structures.**TargetService**

A WSD target service is an abstract entity that can be discovered on the network. Each WSD device must not impersonate more than one target service, even if it hosts multiple services like printing, scanning, etc.

1.2 Transfer

class PyWSD.wsd_transfer__structures.**HostedService**

An actual service offered by a certain wsd target. Each device is a target, but a target can publish multiple hosted services at once.

class PyWSD.wsd_transfer__structures.**TargetInfo**

Holds information about a certain target service, such as name, model, manufacturer, and so on.

1.3 Scan

CHAPTER 2

Operations

2.1 Discovery

`PyWSD.wsd_discovery__operations.`**`get_devices`**(*cache: bool = True, discovery: bool = True, probe_timeout: int = 3, type_filter: Set[str] = None*) → Set[PyWSD.wsd_discovery__structures.TargetService]

Get a list of available wsd-enabled devices

Parameters

- **cache** (*bool*) – True if you want to use the database pointed by *WSD_CACHE_PATH* env variable as a way to know about already discovered devices or not.
- **discovery** (*bool*) – True if you want to rely on multicast probe for device discovery.
- **probe_timeout** (*int*) – the amount of seconds to wait for a probe response
- **type_filter** (*{str}*) – a set of device types (as strings)

Returns a list of wsd targets as TargetService instances

Return type {wsd_discovery__structures.TargetService}

`PyWSD.wsd_discovery__operations.`**`listen_multicast_announcements`**(*sockets: List[socket.socket]*) → Tuple[bool, PyWSD.wsd_discovery__structures.TargetS

`PyWSD.wsd_discovery__operations.`**`read_discovery_multicast_reply`**(*sock: socket.socket, target_service: PyWSD.wsd_discovery__structures.TargetS* → Union[None, Tuple[bool, List[PyWSD.wsd_discovery__structures.

Waits for a reply from an endpoint, containing info about the target itself. Used to catch wsd_probe and wsd_resolve responses. Updates the target_service with data collected.

Parameters

- **sock** (*socket.socket*) – The socket to read from
- **target_service** – an instance of TargetService to fill or update with data received

Returns an updated target_service object, or False if the socket timeout is reached

Return type wsd_discovery__structures.TargetService | False

`PyWSD.wsd_discovery__operations.`**`send_multicast_soap_msg`**(*xml_template: str, fields_map: Dict[str, str], timeout: int*) → socket.socket

Send a wsd xml/soap multicast request, and return the opened socket.

Parameters

- **xml_template** (*str*) – the name of the xml template to fill and send
- **fields_map** (*{str: str}*) – the map of placeholders and strings to substitute inside the template
- **timeout** (*int*) – the timeout of the socket

Returns the socket use for message delivery

Return type socket.socket

`PyWSD.wsd_discovery__operations.`**`wsd_probe`**(*probe_timeout: int = 3, type_filter: Set[str] = None*) → Set[PyWSD.wsd_discovery__structures.TargetService]

Send a multicast discovery probe message, and wait for wsd-enabled devices to respond.

Parameters

- **probe_timeout** (*int*) – the number of seconds to wait for probe replies
- **type_filter** (*{str}*) – a set of legal strings, each representing a device class

Returns a set of wsd targets

Return type {wsd_discovery__structures.TargetService}

`PyWSD.wsd_discovery__operations.`**`wsd_resolve`**(*target_service: PyWSD.wsd_discovery__structures.TargetService*) → Tuple[bool, PyWSD.wsd_discovery__structures.TargetService]

Send a multicast resolve message, and wait for the targeted service to respond.

Parameters **target_service** (*wsd_discovery__structures.TargetService*) – A wsd target to resolve

Returns an updated TargetService with additional information gathered from resolving

Return type *wsd_discovery__structures.TargetService*

2.2 Transfer

`PyWSD.wsd_transfer__operations.`**`wsd_get`**(*target_service: PyWSD.wsd_discovery__structures.TargetService*)

Query wsd target for information about model/device and hosted services.

Parameters **`target_service`** (`wsd_discovery__structures.TargetService`) – A wsd target

Returns A tuple containing a TargetInfo and a list of HostedService instances.

2.3 Eventing

`PyWSD.wsd_eventing__operations.`**`wsd_get_status`**(*hosted_service: PyWSD.wsd_transfer__structures.HostedService, subscription_id: str*) → Union[None, bool, datetime.datetime]

Get the status of an events subscription of a wsd service

Parameters

- **`hosted_service`** (`wsd_transfer__structures.HostedService`) – the wsd service from which you want to hear about the subscription status
- **`subscription_id`** (*`str`*) – the ID returned from a previous successful event subscription call

Returns False if a fault message is received instead, none if the subscription has no expiration set, the expiration date otherwise

Return type None | False | datetime

`PyWSD.wsd_eventing__operations.`**`wsd_renew`**(*hosted_service: PyWSD.wsd_transfer__structures.HostedService, subscription_id: str, expiration: Union[datetime.datetime, datetime.timedelta] = None*) → bool

Renew an events subscription of a wsd service

Parameters

- **`hosted_service`** (`wsd_transfer__structures.HostedService`) – the wsd service that you want to renew the subscription
- **`subscription_id`** (*`str`*) – the ID returned from a previous successful event subscription call
- **`expiration`** (*`datetime | timedelta | None`*) – Expiration time, as a datetime or timedelta object

Returns False if a fault message is received instead, True otherwise

Return type bool

`PyWSD.wsd_eventing__operations.`**`wsd_subscribe`**(*hosted_service: PyWSD.wsd_transfer__structures.HostedService, event_uri: str, notify_addr: str, expiration: Union[datetime.datetime, datetime.timedelta] = None*) → Union[lxml.etree.ElementTree, bool]

Subscribe to a certain type of events of a wsd service

Parameters

- **hosted_service** (`wsd_transfer__structures.HostedService`) – the wsd service to receive event notifications from
- **event_uri** (*str*) – the full URI of the targeted event class. Those URIs are taken from ws specifications
- **notify_addr** (*str*) – The address to send notifications to.
- **expiration** (*datetime | timedelta | None*) – Expiration time, as a datetime or timedelta object

Returns the xml SubscribeResponse of the wsd service or False if a fault message is received instead

Return type lxml.etree.ElementTree | False

`PyWSD.wsd_eventing__operations.`**`wsd_unsubscribe`**(*hosted_service: PyWSD.wsd_transfer__structures.HostedService, subscription_id: str*) → bool

Unsubscribe from events notifications of a wsd service

Parameters

- **hosted_service** (`wsd_transfer__structures.HostedService`) – the wsd service from which you want to unsubscribe for events
- **subscription_id** (*str*) – the ID returned from a previous successful event subscription call

Returns False if a fault message is received instead, True otherwise

Return type bool

2.4 Scan

`PyWSD.wsd_scan__operations.`**`wsd_cancel_job`**(*hosted_scan_service: PyWSD.wsd_transfer__structures.HostedService, job: PyWSD.wsd_scan__structures.ScanJob*) → bool

Submit a CancelJob request, and parse the response. Stops and aborts the specified scan job.

Parameters

- **hosted_scan_service** (`wsd_transfer__structures.HostedService`) – the wsd scan service to query
- **job** (*wsd_scan_structures.ScanJob*) – the ScanJob instance representing the job to abort

Returns True if the job is found and then aborted, False if the specified job do not exists or already ended.

Return type bool

PyWSD.wsd_scan__operations.**wsd_create_scan_job**(*hosted_scan_service: PyWSD.wsd_transfer__structures.HostedService, tkt: PyWSD.wsd_scan__structures.ScanTicket, scan_identifier: str = '', dest_token: str = ''*) → PyWSD.wsd_scan__structures.ScanJob

Submit a CreateScanJob request, and parse the response. This creates a scan job and starts the image(s) acquisition.

Parameters

- **hosted_scan_service** (wsd_transfer__structures.HostedService) – the wsd scan service to query
- **tkt** (*wsd_scan__structures.ScanTicket*) – the ScanTicket to submit for validation purposes
- **scan_identifier** (*str*) – a string identifying the device-initiated scan to handle, if any
- **dest_token** (*str*) – a token assigned by the scanner to this client, needed for device-initiated scans

Returns a ScanJob instance

Return type wsd_scan__structures.ScanJob

PyWSD.wsd_scan__operations.**wsd_get_active_jobs**(*hosted_scan_service: PyWSD.wsd_transfer__structures.HostedService*) → List[PyWSD.wsd_scan__structures.JobSummary]

Submit a GetActiveJobs request, and parse the response. The device should reply with a list of all active scan jobs.

Parameters **hosted_scan_service** (wsd_transfer__structures.HostedService) – the wsd scan service to query

Returns a list of JobSummary elements

Return type list[wsd_scan__structures.JobSummary]

PyWSD.wsd_scan__operations.**wsd_get_job_elements**(*hosted_scan_service: PyWSD.wsd_transfer__structures.HostedService, job: PyWSD.wsd_scan__structures.ScanJob*)

Submit a GetJob request, and parse the response. The device should reply with info's about the specified job, such as its status, the ticket submitted for job initiation, the final parameters set effectively used to scan, and a document list.

Parameters

- **hosted_scan_service** (wsd_transfer__structures.HostedService) – the wsd scan service to query
- **job** (*wsd_scan_structures.ScanJob*) – the ScanJob instance representing the job to abort

Returns a tuple of the form (JobStatus, ScanTicket, DocumentParams, doclist), where doclist is a list of document names

PyWSD.wsd_scan__operations.**wsd_get_job_history**(*hosted_scan_service: PyWSD.wsd_transfer__structures.HostedService*) → List[PyWSD.wsd_scan__structures.JobSummary]

Submit a GetJobHistory request, and parse the response. The device should reply with a list of recently ended jobs. Note that some device implementations do not keep or share job history through WSD.

Parameters **hosted_scan_service** (wsd_transfer__structures.HostedService) – the wsd scan service to query

Returns a list of JobSummary elements.

PyWSD.wsd_scan__operations.**wsd_get_scanner_elements**(*hosted_scan_service: PyWSD.wsd_transfer__structures.HostedService*)

Submit a GetScannerElements request, and parse the response. The device should reply with informations about itself, its configuration, its status and the defalt scan ticket

Parameters **hosted_scan_service** (wsd_transfer__structures.HostedService) – the wsd scan service to query

Returns a tuple of the form (ScannerDescription, ScannerConfiguration, ScannerStatus, ScanTicket)

PyWSD.wsd_scan__operations.**wsd_retrieve_image**(*hosted_scan_service: PyWSD.wsd_transfer__structures.HostedService, job: PyWSD.wsd_scan__structures.ScanJob, docname: str*) → Tuple[int, List[PIL.Image.Image]]

Submit a RetrieveImage request, and parse the response. Retrieves a single image from the scanner, if the job has available images to send. If the file format selected in the scan ticket was multipage, retrieves a batch of images instead. Usually the client has approx. 60 seconds to start images acquisition after the creation of a job.

Parameters

- **hosted_scan_service** (wsd_transfer__structures.HostedService) – the wsd scan service to query
- **job** (*wsd_scan__structures.ScanJob*) – the ScanJob instance representing the queried job.
- **docname** (*str*) – the name assigned to the image to retrieve.

Returns the number of images retrieved, and an array of images

Return type (int, list[PIL.Image])

PyWSD.wsd_scan__operations.**wsd_validate_scan_ticket**(*hosted_scan_service: PyWSD.wsd_transfer__structures.HostedService, tkt: PyWSD.wsd_scan__structures.ScanTicket*) → Tuple[bool, PyWSD.wsd_scan__structures.ScanTicket]

Submit a ValidateScanTicket request, and parse the response. Scanner devices can validate scan settings/parameters and fix errors if any. It is recommended to always validate a ticket before submitting the actual scan job.

Parameters

- **hosted_scan_service** (wsd_transfer__structures.HostedService) – the wsd scan service to query
- **tkt** (*wsd_scan__structures.ScanTicket*) – the ScanTicket to submit for validation purposes

Returns a tuple of the form (boolean, ScanTicket), where the first field is True if no errors were found during validation, along with the same ticket submitted, or False if errors were found, along with a corrected ticket.

Python Module Index

p

PyWSD.wsd_discovery__operations, 5
PyWSD.wsd_discovery__structures, 3
PyWSD.wsd_eventing__operations, 7
PyWSD.wsd_scan__operations, 8
PyWSD.wsd_scan__structures, 3
PyWSD.wsd_transfer__operations, 7
PyWSD.wsd_transfer__structures, 3

Index

G

`get_devices()` (*in module PyWSD.wsd_discovery__operations*), 5

H

`HostedService` (*class in PyWSD.wsd_transfer__structures*), 3

L

`listen_multicast_announcements()` (*in module PyWSD.wsd_discovery__operations*), 5

P

`PyWSD.wsd_discovery__operations` (*module*), 5
`PyWSD.wsd_discovery__structures` (*module*), 3
`PyWSD.wsd_eventing__operations` (*module*), 7
`PyWSD.wsd_scan__operations` (*module*), 8
`PyWSD.wsd_scan__structures` (*module*), 3
`PyWSD.wsd_transfer__operations` (*module*), 7
`PyWSD.wsd_transfer__structures` (*module*), 3

R

`read_discovery_multicast_reply()` (*in module PyWSD.wsd_discovery__operations*), 5

S

`send_multicast_soap_msg()` (*in module PyWSD.wsd_discovery__operations*), 6

T

`TargetInfo` (*class in PyWSD.wsd_transfer__structures*), 3
`TargetService` (*class in PyWSD.wsd_discovery__structures*), 3

W

`wsd_cancel_job()` (*in module PyWSD.wsd_scan__operations*), 8
`wsd_create_scan_job()` (*in module PyWSD.wsd_scan__operations*), 8
`wsd_get()` (*in module PyWSD.wsd_transfer__operations*), 7
`wsd_get_active_jobs()` (*in module PyWSD.wsd_scan__operations*), 9
`wsd_get_job_elements()` (*in module PyWSD.wsd_scan__operations*), 9
`wsd_get_job_history()` (*in module PyWSD.wsd_scan__operations*), 9
`wsd_get_scanner_elements()` (*in module PyWSD.wsd_scan__operations*), 10
`wsd_get_status()` (*in module PyWSD.wsd_eventing__operations*), 7
`wsd_probe()` (*in module PyWSD.wsd_discovery__operations*), 6
`wsd_renew()` (*in module PyWSD.wsd_eventing__operations*), 7
`wsd_resolve()` (*in module PyWSD.wsd_discovery__operations*), 6
`wsd_retrieve_image()` (*in module PyWSD.wsd_scan__operations*), 10
`wsd_subscribe()` (*in module PyWSD.wsd_eventing__operations*), 7
`wsd_unsubscribe()` (*in module PyWSD.wsd_eventing__operations*), 8
`wsd_validate_scan_ticket()` (*in module PyWSD.wsd_scan__operations*), 10